Binary Graph Neural Networks

Mehdi Bahri, Gaétan Bahl, Stefanos Zafeiriou

Binary Graph Neural Networks

Mehdi Bahri[1] Gaétan Bahl[2,3] Stefanos Zafeiriou[1]
[1]Imperial College London, UK [2]Université Côte d'Azur - Inria [3]IRT Saint-Exupéry
{m.bahri, s.zafeiriou}@imperial.ac.uk, gaetan.bahl@inria.fr

Abstract

Graph Neural Networks (GNNs) have emerged as a powerful and flexible framework for representation learning on irregular data. As they generalize the operations of classical CNNs on grids to arbitrary topologies, GNNs also bring much of the implementation challenges of their Euclidean counterparts. Model size, memory footprint, and energy consumption are common concerns for many real-world applications. Network binarization allocates a single bit to parameters and activations, thus dramatically reducing the memory requirements (up to 32x compared to single-precision floating-point numbers) and maximizing the benefits of fast SIMD instructions on modern hardware for measurable speedups. However, in spite of the large body of work on binarization for classical CNNs, this area remains largely unexplored in geometric deep learning. In this paper, we present and evaluate different strategies for the binarization of graph neural networks. We show that through careful design of the models, and control of the training process, binary graph neural networks can be trained at only a moderate cost in accuracy on challenging benchmarks. In particular, we present the first dynamic graph neural network in Hamming space, able to leverage efficient k-NN search on binary vectors to speed-up the construction of the dynamic graph. We further verify that the binary models offer significant savings on embedded devices. Our code is publicly available on Github[1].

1. Introduction

Standard CNNs assume their input to have a regular grid structure, and are therefore suitable for data that can be well-represented in an Euclidean space, such as images, sound, or videos. However, many increasingly relevant types of data do not fit this framework [5]. Graph theory offers a broad mathematical formalism for modeling interactions, and is therefore commonly used in fields such as network sciences [12], bioinformatics [24, 40], and recommender systems [37], as well as for studying discretisations of continuous mathematical structures such as in computer graphics [4]. This motivates the development of machine learning methods able to deal with graph-supported data. Among them, Graph Neural Networks (GNNs) generalize the operations of CNNs to arbitrary topologies by extending the basic building blocks of CNNs such as convolutions and pooling to graphs. Similarly to CNNs, GNNs learn deep representations of graphs or graph elements, and have emerged as the best performing models for learning on graphs as well as on 3D meshes with the development of advanced and increasingly deep architectures [33, 18].

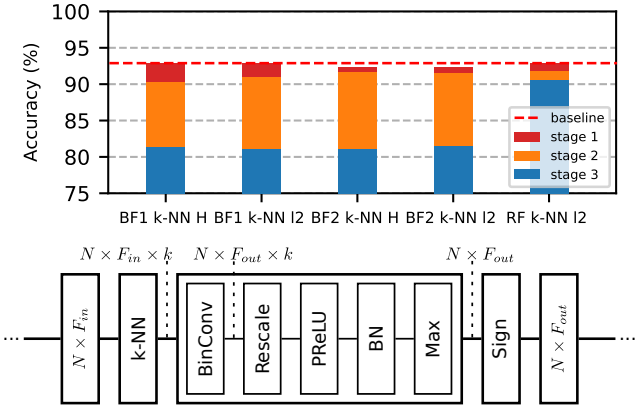

Figure 1. **Top:** Test accuracy of different binarization schemes at all stages of our cascaded distillation protocol (baseline: 92.89%). **Bottom:** The "BF2" variant of our XorEdgeConv operator.

As the computational complexity of the networks and the scale of graph datasets increase, so does the need for faster and smaller models. The motivations for resource-efficient deep learning are numerous and also apply to deep learning on graphs and 3D shapes. Computer vision models are routinely deployed on embedded devices, such as mobile phones or satellites [2, 32], where energy and storage constraints are important. The development of smart devices and IoT may bring about the need for power-efficient graph learning models [27, 62, 8]. Finally, models that require GPUs for inference can be expensive to serve, whereas CPUs are typically more affordable. This latter point is especially relevant

[1] https://github.com/mbahri/binary_gnn

to the applications of GNNs in large-scale data mining on relational datasets, such as those produced by popular social networks, or in bioinformatics [35].

While recent work has proposed algorithmic changes to make graph neural networks more scalable, such as the use of sampling [21, 61] or architectural improvements [15, 10] and simplifications [56], our approach is orthogonal to these advances and focuses on compressing existing architectures while preserving model performance. Model compression is a well-researched area for Euclidean neural networks, but has seen very little application in geometric deep learning. In this paper, we study different strategies for binarizing GNNs.

Our contributions are as follows:

- We present a binarization strategy inspired by the latest developments in binary neural networks for images [7, 36] and knowledge distillation for graph networks
- We develop an efficient dynamic graph neural network model that constructs the dynamic graph in Hamming space, thus paving the way for significant speedups at inference time, with negligible loss of accuracy when using real-valued weights
- We conduct a thorough ablation study of the hyperparameters and techniques used in our approach
- We demonstrate real-world acceleration of our models on a budget ARM device

Notations Matrices and vectors are denoted by upper and lowercase bold letters (*e.g.*, $\mathbf{X}$ and $\mathbf{x}$), respectively. $\mathbf{I}$ denotes the identity matrix of compatible dimensions. The i^{th} column of $\mathbf{X}$ is denoted as $\mathbf{x}_i$. The set of real numbers is denoted by $\mathbb{R}$. A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of *vertices* $\mathcal{V} = \{1, \dots, n\}$ and *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The *neighborhood* of vertex i, denoted by $\mathcal{N}(i) = \{j : (i, j) \in \mathcal{E}\}$, is the set of vertices adjacent to i. Other mathematical notations are summarized in Appendix E of the Supplementary Material.

2. Related Work

Knowledge distillation uses a pretrained *teacher* network to supervise and inform the training of a smaller *student* network . In logit matching [22], a cross-entropy loss is used to regularize the output logits of the student by matching them with a blurred version of the teacher's logits computed using a softmax with an additional temperature hyperparameter. More recent works also focus on matching internal activations of both networks, such as attention volumes in [60], or on preserving relational knowledge [44, 31, 49].

Quantized and Binary Neural Networks Network quantization [19, 63] refers to the practice of lowering the numerical precision of a model in a bid to reduce its size and speed-up inference. Binary Neural Networks (BNNs) [25] push it to the extreme and use a single bit for weights and activations. The seminal work of XNOR-Net [47] showed that re-introducing a small number of floating point operations in BNNs can drastically improve the performance compared to using pure binary operations by reducing the quantization error. In XNOR-Net, a dot product $\star$ between real vectors $\mathbf{a}$ and $\mathbf{b}$ of dimension n is approximated by $\mathbf{a} \star \mathbf{b} \approx (\text{sign}(\mathbf{a}) \circledast \text{sign}(\mathbf{b}))\alpha\beta$, where $\beta = \frac{1}{n}||\mathbf{a}||_1$ and $\alpha = \frac{1}{n}||\mathbf{b}||_1$ are rescaling constants. XNOR-Net++ [7] proposed to instead learn a rescaling tensor Γ, with shared factors to limit the number of trainable parameters and avoid overfitting. Finally, in Real-to-Binary networks [36], the authors compile state-of-the-art techniques and improve the performance of binary models with knowledge distillation.

Graph Neural Networks Graph Neural Networks were initially proposed in [20, 48] as a form of recursive neural networks. Later formulations relied on Fourier analysis on graphs using the eigendecomposition of the graph Laplacian [6] and approximations of such [11], but suffered from the connectivity-specific nature of the Laplacian. Attention-based models [38, 14, 51, 50] are purely spatial approaches that compute a vertex's features as a dynamic weighting of its neighbours'. Spatial and spectral approaches have been unified [29] and shown to derive from the more general neural message passing [17] framework. We refer to recent reviews on GNNs, such as [57], for a comprehensive overview, and focus only on the operators we binarize in this paper.

The message-passing framework offers a general formulation of graph neural networks:

$$\mathbf{x}_i^{(l)} = \gamma^{(l)}\left(\mathbf{x}_i^{(l-1)}, \underset{j \in \mathcal{N}(i)}{\square} \phi^{(l)}\left(\mathbf{x}_i^{(l-1)}, \mathbf{x}_j^{(l-1)}, \mathbf{e}_{ij}^{(l-1)}\right)\right), \tag{1}$$

where $\square$ denotes a differentiable symmetric (permutation-invariant) function, (*e.g.* $\max$ or $\sum$), ϕ a differentiable kernel function, γ is an MLP, and $\mathbf{x}_i$ and $\mathbf{e}_{ij}$ are features associated with vertex i and edge (i, j), respectively.

The EdgeConv operator is a special case introduced as part of the Dynamic Graph CNN (DGCNN) model [54] and defines an edge message as a function of $\mathbf{x}_j - \mathbf{x}_i$:

$$\mathbf{e}_{ij}^{(l)} = \text{ReLU}\left(\boldsymbol{\theta}^{(l)}(\mathbf{x}_j^{(l-1)} - \mathbf{x}_i^{(l-1)}) + \boldsymbol{\phi}^{(l)}\mathbf{x}_i^{(l-1)}\right) \tag{2}$$

$$= \text{ReLU}\left(\boldsymbol{\Theta}^{(l)}\tilde{\mathbf{X}}^{(l-1)}\right) \tag{3}$$

where $\tilde{\mathbf{X}}^{(l-1)} = \left[\mathbf{x}_i^{(l-1)} \,||\, \mathbf{x}_j^{(l-1)} - \mathbf{x}_i^{(l-1)}\right]$, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are trainable weights, and $\boldsymbol{\Theta}$ their concatenation.

The output of the convolution is the max aggregation ($\square = \max$) of the edge messages:

$$\mathbf{x}_i^{(l)} = \max_{j \in \mathcal{N}(i)} \mathbf{e}_{ij}^{(l)} \tag{4}$$

While the EdgeConv operator is applicable to graph inputs, the main use case presented in [54] is for point clouds, where the neighbours are found by k-Nearest Neighbours (k-NN) search in feature space before each convolutional layer. DGCNN is the first example of a dynamic graph architecture, with follow-up work in [26].

The GraphSAGE [21] operator introduced inductive learning on large graphs with sampling and can also be phrased as a message passing operator:

$$\mathbf{x}_i^{(l)} = \mathrm{Norm}\left(\mathrm{ReLU}\left(\mathbf{W}^{(l)}\left[\mathbf{x}_i^{(l-1)} \,||\, \underset{j\in\mathcal{N}(i)}{\mathrm{Aggr}}\, \mathbf{x}_j^{(l-1)}\right]\right)\right) \tag{5}$$

Where Aggr is a symmetric aggregation function such as $\max$, sum or mean; Norm denotes the ℓ_2 normalization, and $\mathbf{W}$ is a tensor of learnable weights.

Model Compression in Geometric Deep Learning In [52], the authors propose to binarize the Graph Attention (GAT) operator [50], and evaluate their method on small-scale datasets such as Cora [39] and Pubmed [29]. In [53], the authors apply the XNOR-Net approach to GCN [29] with success, but also on small-scale datasets. Finally, [46] propose to binarize PointNet with tailored aggregation and scaling functions. At the time of writing, the Local Structure Preserving (LSP) module of [59] is the only knowledge distillation method specifically designed for GNNs. LSP defines local structure vectors LS_i for each node in the graph:

$$LS_{ij} = \frac{\exp(\mathrm{SIM}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k\in\mathcal{N}(I)} \exp(\mathrm{SIM}(\mathbf{x}_i, \mathbf{x}_k))} \tag{6}$$

where SIM denotes a similarity measure, *e.g.*, $||.||_2^2$ or a kernel function such as a Gaussian RBF kernel. The total local structure preserving loss between a student network s and a teacher t is then defined as:

$$L_{LSP} = \frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}^u(i)} LS_{ij}^s \log\frac{LS_{ij}^s}{LS_{ij}^t}. \tag{7}$$

$\mathcal{N}^u(i) = \mathcal{N}^s(i) \cup \mathcal{N}^t(i)$ to support dynamic graph models.

3. Method

Eq. 1 is more general than the vanilla Euclidean convolution, which boils down to a single matrix product to quantize. We must therefore choose which elements of Eq. 1 to binarize and how: the node features $\mathbf{x}_i$, the edge messages $\mathbf{e}_{ij}$, and the functions $\square$, γ and ϕ may all need to be adapted.

Quantization We follow the literature and adopt the sign operator as the binarization function:

$$\mathrm{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}. \tag{8}$$

As the gradient of sign is zero almost everywhere, we employ the straight-through estimator [3] to provide a valid gradient. We use this method for both network weights and activations. Furthermore, we mean-center and clip the real latent network weights after their update in the backpropagation step.

Learnable rescaling Assuming a dot product operation (*e.g.* a fully-connected or convolutional layer) $\mathbf{A} \star \mathbf{B} \in \mathbb{R}^{o\times h\times w}$, we approximate it as in [7]:

$$\mathbf{A} \star \mathbf{B} \approx (\mathrm{sign}(\mathbf{A}) \circledast \mathrm{sign}(\mathbf{B})) \odot \Gamma, \tag{9}$$

with Γ a learned rescaling tensor. We use two constructions of Γ depending on the model. Channel-wise:

$$\Gamma = \alpha \in \mathbb{R}^{o\times 1\times 1} \tag{10}$$

and one rank-1 factor per mode:

$$\Gamma = \alpha \otimes \beta \otimes \gamma, \quad \alpha \in \mathbb{R}^o, \beta \in \mathbb{R}^h, \gamma \in \mathbb{R}^w \tag{11}$$

Activation functions Recent work [36] has shown using non-linear activations in XNOR-Net - type blocks can improve the performance of binary neural networks, with PReLU bringing the most improvement.

Knowledge Distillation Inspired by [36], we investigate the applicability of knowledge distillation for the binarization of graph neural networks. For classification tasks, we use a logit matching loss [22] as the base distillation method. We also implemented the LSP module of [59].

Multi-stage training We employ a cascaded distillation scheme [36], an overview of which is shown in Figure 2.
Stage 1: We first build a real-valued and real-weighted network with the same architecture as the desired binary network by replacing the quantization function with $\tanh$. We distillate the original (base) network into this first student network. We employ weight decay with weight $\lambda = 1e-5$, logit matching, and LSP. We use the same initial learning rate and learning rate schedule as for the base network.
Stage 2: The model of stage 1 becomes the teacher, the student is a binary network with real-valued weights but binary activations. We initialize the student with the weights of the teacher. We employ weight decay with $\lambda = 1e-5$, logit matching, and LSP. We use a smaller learning rate (*e.g.* 25%) than for stage 1 and the same learning rate schedule.
Stage 3: The model of stage 2 becomes the teacher, the student is a binary network with binary weights and binary activations. We use logit matching and LSP but no weight decay. The hyperparameters we used are available in Section 5.2. We did not observe a significant difference in models initialized randomly or using the weights of the teacher.

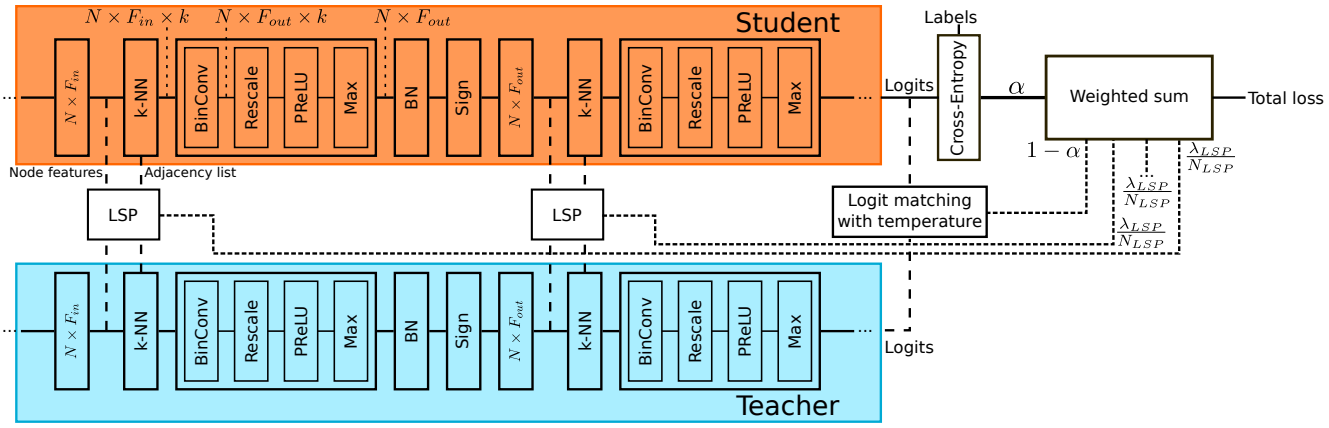

Figure 2. **Distillation with the "BF1" variant of XorEdgeConv:** the student model is more heavily quantized than the teacher. Knowledge transfer points equipped with LSP modules encourage similar dynamic graph feature distributions after each k-NN graph computation (except for the first, performed on the fixed 3D coordinates). Logit matching is used to further inform the training of the student.

Batch Normalization We investigate the importance of the order of the dot product and batch normalization operations for discretizing dot product operations within graph convolution operators. However, our base approach is to follow the XNOR-Net block structure [47] with learnable rescaling (*i.e.* XNOR-Net++ block). In particular, all fully-connected layers of MLPs that follow graph feature extraction layers are binarized using the XNOR-Net++ block.

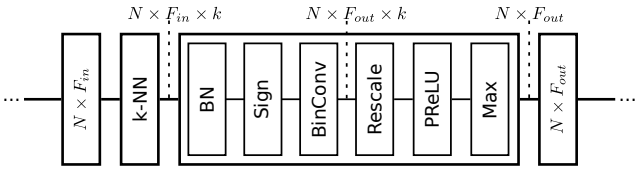

Figure 3. **The BinEdgeConv operator** ("RF" model in the experiments) can be deployed as a drop-in replacement for EdgeConv and follows the XNOR-Net++ approach to binarization.

4. Models

We choose the Dynamic Graph CNN model, built around the EdgeConv operator of Eq. 3 as our main case study. DGCNN has several characteristics that make it an interesting candidate for binarization. First, the EdgeConv operator is widely applicable to graphs and point clouds. Second, the operator relies on both node features and edge messages, contrary to other operators previously studied in GNN binarization such as GCN. Third, the time complexity of DGCNN is strongly impacted by the k-NN search in feature space. k-NN search can be made extremely efficient in Hamming space, and fast algorithms could theoretically be implemented for the construction of the dynamic graph at inference, provided that the graph features used in the search are binary, which requires a different binarization strategy than merely approximating the dense layer in EdgeConv.

For completeness, we also derive a binary SAGE operator.

4.1. Direct binarization

Our first approach binarizes the network weights and the graph features at the input of the graph convolution layers, but keeps the output real-valued. The network, therefore, produces real-valued node features. We replace the EdgeConv operator by a block similar to XNOR-Net++, using learnable rescaling and batch normalization pre-quantization.

We define the BinEdgeConv operator as:

$$\mathbf{e}_{ij}^{(l)} = \sigma\left(\text{sign}(\mathbf{\Theta}^{(l)}) \circledast \text{sign}\left(\text{BN}\left(\tilde{\mathbf{X}}^{(l-1)}\right)\right) \odot \Gamma^{(l)}\right) \tag{12}$$

$$\mathbf{x}_i^{(l)} = \max_{j \in \mathcal{N}(i)} \mathbf{e}_{ij}^{(l)} \tag{13}$$

with σ the PReLU activation, and $\Gamma^{(l)}$ a real rescaling tensor. BinEdgeConv is visualized in Figure 3.

We use the same structure to approximate the MLP classifier. Similarly, we binarize Eq. 5 to get:

$$\mathbf{h}^{(l)} = \text{sign}\left(\text{BN}\left(\left[\mathbf{x}_i^{(l-1)} \,||\, \underset{j \in \mathcal{N}(i)}{\text{Aggr}}\, \mathbf{x}_j^{(l-1)}\right]\right)\right) \tag{14}$$

$$\mathbf{x}_i^{(l)} = \text{Norm}\left(\sigma\left(\left(\text{sign}(\mathbf{W}^{(l)}) \circledast \mathbf{h}^{(l)}\right) \odot \Gamma^{(l)}\right)\right). \tag{15}$$

with σ the PReLu activation and $\Gamma^{(l)}$ following Eq. 10.

4.2. Dynamic Graph in Hamming Space

As mentioned, one advantage of binary node features is to enable fast computation of the k-Nearest Neighbours graph at inference time by replacing the ℓ_2 norm with the Hamming distance. We detail our approach to enable quantization-aware training with k-NN search on binary vectors.

Edge feature The central learnable operation of EdgeConv is $\mathbf{\Theta}\left[\mathbf{x}_i \,||\, \mathbf{x}_j - \mathbf{x}_i\right]$ as per Eq. 3, where the edge feature is $\mathbf{x}_j - \mathbf{x}_i$. Assuming binary node features, the standard subtraction operation becomes meaningless. Formally, for $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ with $\mathbb{R}^n$ the n-dimensional Euclidean vector space over the field of real numbers,

$$\mathbf{x}_1 - \mathbf{x}_2 \coloneqq \mathbf{x}_1 + (-\mathbf{x}_2) \tag{16}$$

by definition, with $(-\mathbf{x}_2)$ the additive inverse of $\mathbf{x}_2$. Seeing binary vectors as elements of vector spaces over the finite field $\mathbb{F}_2$, we can adapt Eq. 16 with the operations of boolean algebra. The addition therefore becomes the boolean exclusive or (XOR) $\oplus$, and the additive inverse of $(-x)_{\mathbb{F}_2}$ is x itself ($x \oplus x = 0$). With our choice of quantizer (Eq. 8), $\mathbf{x}_i, \mathbf{x}_j \in \{-1, 1\}^n$ and we observe that $\mathbf{x}_i \oplus \mathbf{x}_j = -\mathbf{x}_i \odot \mathbf{x}_j$. We therefore base our binary EdgeConv operator for binary node features, XorEdgeConv, on the following steps:

$$\mathbf{e}_{ij}^{(l)} = \sigma\left(\text{sign}(\mathbf{\Theta}^{(l)}) \circledast \tilde{\mathbf{X}}_b^{(l-1)} \odot \Gamma^{(l)}\right) \tag{17}$$

$$\mathbf{x}_i^{(l)} = \text{sign}\left(\max_{j \in \mathcal{N}(i)} \mathbf{e}_{ij}^{(l)}\right) \tag{18}$$

with $\tilde{\mathbf{X}}_b^{(l-1)} = \left[\mathbf{x}_i^{(l-1)} \,||\, -\mathbf{x}_j^{(l-1)} \odot \mathbf{x}_i^{(l-1)}\right]$, $\mathbf{\Theta}^{(l)}$ a set of learnable real parameters and $\Gamma^{(l)}$ a real rescaling tensor. We further investigate the practical importance of the placement of the batch normalization operation, either before or after the aggregation function, by proposing two variants:

$$\mathbf{x}_i^{(l)} = \text{sign}\left(\text{BN}\left(\max_{j \in \mathcal{N}(i)} \mathbf{e}_{ij}^{(l)}\right)\right) \tag{19}$$

shown as part of Figure 2 and

$$\mathbf{x}_i^{(l)} = \text{sign}\left(\max_{j \in \mathcal{N}(i)} \text{BN}\left(\mathbf{e}_{ij}^{(l)}\right)\right) \tag{20}$$

drawn in Figure 1. Here, the main difference lies in the distribution of the features pre-quantization.

Nearest Neighbours Search The Hamming distance between two binary vectors $\mathbf{x}, \mathbf{y}$ is $d_H(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} \oplus \mathbf{y}||_H$ where $||.||_H$ is the number of non-zero bits, and can be efficiently implemented as `popcount(x xor y)`. We note that this relates our approach to previous work on efficient hashing [41, 43, 30] and metric learning [42], especially given the dynamic nature of the graph. Unfortunately, like the sign function, the hamming distance has an ill-defined gradient, which hinders its use as-is for training. We therefore investigate two continuous relaxations. (1) we use the standard ℓ_2 norm for training, since all norms are equivalent in finite dimensions. (2) we observe that the matrix of pairwise Hamming distances between d-dimensional vectors $\mathbf{x}_i$ valued in $\{-1, 1\}$ can be computed in a single matrix-matrix product up to a factor 2 as (see Eq. 5 of [34]):

$$\mathbf{D} = -(\mathbf{X}\mathbf{X}^T - d\mathbf{I}_d) \tag{21}$$

with $\mathbf{X}$ the matrix of the $\mathbf{x}_i$ stacked row-wise, and $\mathbf{I}_d$ the identity matrix. We investigate both options.

Local structure With binary node features, we now have to choose how to define the local structure similarity measure of Eq. 6. One option is to use the standard Gaussian RBF as in the real-valued case. Another option is to define the similarity in Hamming space, like for the k-NN search. We therefore investigate the following similarity metric:

$$\text{SIM}(\mathbf{x}_i, \mathbf{x}_j) = e^{-||\mathbf{x}_i \oplus \mathbf{x}_j||_H} \tag{22}$$

For vectors $\mathbf{x}, \mathbf{y} \in \{-1, 1\}^n$, we note that $||\mathbf{x} \oplus \mathbf{y}||_H = \frac{1}{2}\sum_{k=1}^{n}(-x_k y_k + 1)$.

5. Experimental Evaluation

We perform a thorough ablation study of our method on Dynamic Graph CNN. The model binarized according to the method of Section 4.1 and using the BinEdgeConv operator of Eq. 12 is referred to as **RF** for "Real graph Features". The model binarized according to Section 4.2 and using the XorEdgeConv operator is referred to as **BF1**, if following Eq. 19, or **BF2**, if following Eq. 20. We evaluate DGCNN on the ModelNet40 classification benchmark, as in [54]. We implement Γ as per Eq. 11 for the **RF** model and Eq. 10 for the **BF** models. In the next paragraphs, the numbers in parentheses refer to the corresponding lines of Table 1.

Balance functions (16-18, 23-24) Recent work [46] has uncovered possible limitations in binary graph and point cloud learning models when quantizing the output of the max-pooling aggregation of batch-normalized high-dimensional features. Similarly, the authors of [52] claim that a balance function is necessary to avoid large values in the outputs of the dot product operations when most pre-quantization inputs are positive.

We evaluate two strategies for re-centering the input of sign , namely mean-centering, and median-centering (thus ensuring a perfectly balanced distribution of positive and negative values pre-quantization). We evaluate these techniques for the max aggregation of edge messages ("edge balance", *e.g.* between the framed block and the sign operation in Figure 1) and for the max and average pooling operations before the MLP classifier ("global balance").

We can see in Table 1 that in all cases, the addition of balance functions actually lowered the performance of the models. This suggests that using batch normalization prior to quantization, as is common in the binary CNN literature, is sufficient at the message aggregation level and for producing graph embedding vectors.

#	Model	Distillation	Stage	k-NN	LSP	λ_{LSP}	Activation	Global balance	Edge balance	Acc (%)
1	BF1	None	-	H	-	-	PReLU	-	-	61.26
2	BF1	Direct	-	H	-	-	PReLU	-	-	62.84
3	BF1	Cascade	3	H	-	-	PReLU	-	-	**81.04**
4	BF1	Cascade	3	H	-	-	None	-	-	74.80
5	BF1	Cascade	2	H	ℓ_2	100	PReLU	-	-	*90.32*
6	BF1	Cascade	3	H	ℓ_2	100	PReLU	-	-	**81.32**
7	BF1	Cascade	3	H	ℓ_2	100	ReLU	-	-	76.46
8	BF1	None	-	ℓ_2	-	-	PReLU	-	-	62.97
9	BF1	Direct	-	ℓ_2	-	-	PReLU	-	-	61.95
10	BF1	Cascade	3	ℓ_2	-	-	PReLU	-	-	**81.00**
11	BF1	Cascade	2	ℓ_2	ℓ_2	100	PReLU	-	-	*91.00*
12	BF1	Cascade	3	ℓ_2	ℓ_2	100	PReLU	-	-	**81.00**
13	BF2	None	-	H	-	-	PReLU	-	-	59.72
14	BF2	Direct	-	H	-	-	PReLU	-	-	59.00
15	BF2	Cascade	3	H	-	-	PReLU	-	-	**79.58**
16	BF2	Cascade	3	H	-	-	PReLU	-	Mean	51.05
17	BF2	Cascade	3	H	-	-	PReLU	-	Median	**75.93**
18	BF2	Cascade	3	H	-	-	None	-	Median	71.96
19	BF2	Cascade	2	H	ℓ_2	100	PReLU	-	-	*91.57*
20	BF2	Cascade	3	H	ℓ_2	100	PReLU	-	-	**81.08**
21	BF2	Cascade	3	H	ℓ_2	100	None	-	-	76.09
22	BF2	Cascade	3	H	ℓ_2	100	ReLU	-	-	**76.22**
23	BF2	Cascade	3	H	ℓ_2	100	PReLU	Mean	-	**67.87**
24	BF2	Cascade	3	H	ℓ_2	100	PReLU	Median	-	60.82
25	BF2	None	-	ℓ_2	-	-	PReLU	-	-	57.90
26	BF2	Direct	-	ℓ_2	-	-	PReLU	-	-	59.12
27	BF2	Cascade	3	ℓ_2	-	-	PReLU	-	-	**80.11**
28	BF2	Cascade	2	ℓ_2	ℓ_2	100	PReLU	-	-	*91.53*
29	BF2	Cascade	3	ℓ_2	ℓ_2	100	PReLU	-	-	**81.52**
30	RF	None	-	ℓ_2	-	-	PReLU	-	-	79.30
31	RF	Direct	-	ℓ_2	-	-	PReLU	-	-	72.69
32	RF	Cascade	3	ℓ_2	-	-	PReLU	-	-	**91.05**
33	RF	Cascade	3	ℓ_2	ℓ_2	100	PReLU	-	-	**90.52**
34	RF	Cascade	3	ℓ_2	ℓ_2	100	None	-	-	89.71
35	RF	Cascade	3	ℓ_2	ℓ_2	100	ReLU	-	-	89.59
36	Baseline	-	-	-	-	ReLU	-	-	-	92.89

Table 1. Different variants and ablations of our binarized DGCNN models on the ModelNet40 dataset.

Non-linear activation (3-4,6-7,17-18,20-22,33-35) Since the sign operation can be seen as acting as an activation applied on the output and to the weights of the XorEdgeConv operator, we compare the models with binary node features with PReLU, ReLU, or no additional activation in Table 1. We can see the PReLU non-linearity offers significant improvements over the models trained with ReLU or without non-linearity in the edge messages, at the cost of a single additional `fp32` parameter - the largest improvement being observed for the models that apply either median-centering or batch normalization before the quantization operation.

Binary node features and k-NN We now study the final performance of our models depending on whether we use BinEdgeConv (real node features) or XorEdgeConv. Looking at the final models (stage 3) in Table 1, the model with real-valued node features that performs k-NN search with the ℓ_2 norm (32) performs comparably with the full floating-point model (36). On the other hand, we saw a greater reduction in accuracy with the binary node features for the full binary models (6,12,20,29), and comparable accuracy whether we use the ℓ_2 norm (12,29) or the relaxed Hamming distance (6,20). However, as reported in Table 1, using real weights (stage 2) with binary node features and k-NN search performed in Hamming space (5,11,19,28) matched the performance of the original floating point model (36).

We found stage 3 to be crucial to the final model's performance, and sensitive to the choice of learning rate and learning rate schedule. This suggests that, although more research and parameter tuning is necessary to maximize the performance of the full binary networks in Hamming space, dynamic graph networks that learn binary codes and con-

struct the dynamic graph in Hamming space can be trained with minimal reduction in performance.

Impact of LSP The node features of the teacher and of the students are always real-valued at stage 1. Stage 2 was carried out using either the Gaussian RBF similarity or Eq. 22 for the student (which may have binary node features) and the Gaussian RBF for the teacher. Stage 3 uses either similarity measure for both the teacher and student. We also report the results of distilling the baseline DGCNN (full floating-point) model into a BF1 or BF2 full-binary model using the similarity in Hamming space for the student.

We saw inconsistent improvements when using LSP with the Gaussian RBF (ℓ_2), as seen in Table 1 (3,6,10,12,15,20,27,29,32,33). This suggest the usefulness of the additional structure preserving knowledge is situational, as it can both increase (3,4,15,20,27,29) or decrease model performance (32,33). Contrary to the models trained using k-NN search performed in Hamming space, the models trained with distillation using Eq. 22 did not match the performance of their Gaussian ℓ_2 counterparts, as shown in Table 2, which we conjecture to be due to poor gradients.

Model	Stage	KNN	LSP	λ_{LSP}	Acc. (%)
BF1	2	H	H	100	38.21
BF1	2	ℓ_2	H	100	38.94
BF2	2	H	H	100	63.25
BF2	2	ℓ_2	H	100	64.71
BF1	3	H	H	100	16.29
BF1	3	ℓ_2	H	100	20.34
BF2	3	ℓ_2	H	100	9.40
BF2	3	H	H	100	11.47
BF1	Direct	ℓ_2	H	100	23.34
BF2	Direct	H	H	100	30.23
BF2	Direct	ℓ_2	H	100	32.17
BF1	Direct	H	H	100	36.47

Table 2. Performance of models trained with LSP using the Hamming-based similarity of Eq. 22 (H) at different stages and for direct distillation. Compared to the models trained using the Gaussian RBF (ℓ_2) similarity, low performance was observed.

Cascaded distillation (1-3,13-15,25-27,30-32) Table 1 compares distilling the baseline network directly into a binary network, training from scratch, and the three-stage distillation. We observed consistently higher performance with the progressive distillation, confirming its effectiveness.

Large-scale inductive learning with GraphSAGE We benchmark our binarized GraphSAGE on the OGB-Products and OGB-Proteins node property prediction datasets [23], which are recent and challenging (2,449,029 nodes, 61,859,140 edges for OGB-Product) benchmarks with standardized evaluation procedures, compared to the more commonly used ones, such as Cora [39] (2708 nodes, 5429 edges) used in [52] or Reddit [21] (232,965 nodes, 114,615,892 edges) used in [53]. Notably, the Proteins dataset is challenging due to the high average node degree and small graph diameter, which may exacerbate limitations of GNNs [1].

	OGBN-Products		OGBN-Proteins	
Model	Mean acc.	Std. acc.	Mean acc.	Std. acc.
SAGE fp32	0.7862	0.0043	0.7734	0.0041
SAGE bin f.s.	0.7300	0.0156	0.7497	0.0047
SAGE bin l.m.	0.7260	0.0153	-	-
GCN fp32	0.7564	0.0021	0.7254	0.0044

Table 3. Final test accuracy on the OGB-Products and OGB-Proteins node property prediction benchmarks, averaged over 10 runs. "f.s.": from scratch. "l.m.": logit matching.

We implemented BinSAGE according to Eq. 15 - details of the architecture can be found in Section 5.2. For OGB-Products, we use logit matching only for distillation and no PReLU activation. For OGB-Proteins, we use PReLU activations and no distillation, as the task is multi-label classification, and the very large number of edges made using LSP impractical. We use channel-wise rescaling only for both to maximize scalability. On OGB-Products, we did not observe a statistically significant different between training the model from scratch and three-stage distillation with logit matching: in both cases, the full binary model came within 5-6% of the full-precision model. On OGB-Proteins, the simple binary network trained from scratch is within 3% of the accuracy of the full-precision network and outperforms the full-precision GCN. This suggests other strategies to improve model scalability, in this case sampling, can be successfully combined with our binarisation method.

5.1. Speed on embedded hardware

In order to measure the speed improvements yielded by our binary conversion scheme, we benchmark it on a Raspberry Pi 4B board with 4GB of RAM and a Broadcom BCM2711 Quad core Cortex-A72 (ARM v8) 64-bit SoC clocked at 1.5GHz, running Manjaro 64-bit. The Pi is a popular, cheap, and readily available ARM-based platform, and is thus a good fit for our experiments.

We benchmark four DGCNN models, in order to measure the speedup for each subsequent optimization. The specifics of each model are given in Table 4. The input size is set to 1024 points with a batch size of 8, 40 output classes, and 20 nearest neighbors. We convert our models to Tensorflow Lite using LARQ [16], an open-source library for binarized neural networks, and benchmark them using the LARQ Compute Engine (LCE) tool. Once converted, the smallest model's file size is only 341KB down from 7.2MB, for a 20x reduction.

Figure 4 shows the benchmark results. Our optimized binary model halves the run-time, thus achieving a substantial speedup. Peak memory usage is also significantly reduced,

Model	Binary Weights	Binary Features	Hamming Distance
DGCNN			
BDGCNN RF	✓		
BDGCNN BF	✓	✓	
BDGCNN BF H	✓	✓	✓

Table 4. Features of benchmarked models. Hamm Dist = Pairwise Hamming distance instead of ℓ_2, implemented in ARM NEON operations on bit-packed features (simulated).

from 575MB to 346MB. It is to be noted that DGCNN is a complex model with costly operations, such as concatenation and top-k, that are not made faster by binarization (denoted as "incompressible ops" in Figure 4). We provide a profiling of the models in Appendix B of the Supplementary Material.

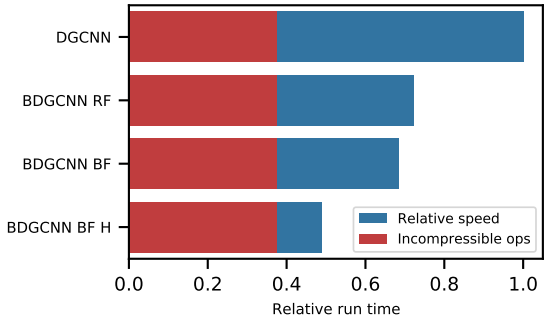

Figure 4. Relative run time on a Raspberry Pi 4B compared to the base DGCNN model. A 2x speedup is achieved by our final optimized model. Run times computed with the LCE benchmark tool over 50 runs.

Unfortunately, we did not have an optimized version of the Hamming distance in LCE at the time of writing. Thus, the final result is simulated by profiling the run-time of an un-optimized implementation, and estimating the savings we would get with ARM NEON instructions. It is theoretically possible to treat 64 features at a time, and achieve a 64x speedup (or higher by grouping loads and writes with `vld4`). We use 32x as a *conservative estimate* since we couldn't account for LCE's bit-packed conversion.

5.2. Implementation details

For DGCNN, we follow the architecture of [55]. For GraphSAGE, we use the baseline architecture of the OGB benchmarks [23]; that is, three layers with 256 hidden features and mean aggregation. We use three knowledge transfer points for LSP on DGCNN, one after each EdgeConv layer except for the first layer (the k-NN and graph features are computed on the 3D coordinates of the point clouds and do not change). All binary models use binary inputs for the convolution and dense layers. For DGCNN, the final layer of the MLP classifier is kept real-weighted, as is customary in the binary neural network literature due to the small number of parameters, but the input features are binarized. For GraphSAGE, all three layers have binary weights.

Our models are implemented in Pytorch [45]. We use the implementation of DGCNN by the authors as a starting point, and Pytorch Geometric [13] for GraphSAGE and OGB [23]. We use the Adam optimizer [28]. We train the DGCNN models for 250 epochs for stage 1 and 350 epochs for stages 2 and 3, on 4 Nvidia 2080 Ti GPUs. The intial learning rate of stage 1 is set to $1e-3$ and for stage 2 to $2.5e-4$, with learning rate decay of 0.5 at 50% and 75% of the total number of epochs. For stage 3, we set the learning rate to $1e-3$ and decay by a factor of 0.5 every 50 epochs. We trained GraphSAGE according to the OGB benchmark methodology, using the provided training, validation, and test sets. We trained all models for 20 epochs and averaged the performance over 10 runs. For GraphSAGE, we used ℓ_2 regularization on the learnable scaling factors only, with a weight $\lambda = 1e-4$. For logit matching, we set $T = 3$ and $\alpha = 1e-1$. For LSP, we set $\lambda_{LSP} = 1e2$.

6. Conclusion

In this work, we introduce a binarization scheme for GNNs based on the XNOR-Net++ methodology and knowledge distillation. We study the impact of various strategies and design decisions on the final performance of binarized graph neural networks, and show that our approach allows us to closely match or equal the performance of floating-point models on difficult benchmarks, with significant reductions in memory consumption and inference time. We further demonstrate that dynamic graph neural networks can be trained to high accuracy with binary node features, enabling fast construction of the dynamic graph at inference time through efficient Hamming-based algorithms, and further relating dynamic graph models to metric learning and compact hashing. Our DGCNN in Hamming space nearly equals the performance of the full floating point model when trained with floating point weights, and offers competitive accuracy with large speed and memory savings when trained with binary weights. We believe higher performance can already be obtained with this model by adjusting the learning rate schedule in the final distillation stage. Future work will investigate further improving the accuracy of the models, theoretical properties on binary graph convolutions, and inference with fast k-NN search in Hamming space.

Acknowledgements M. B. is supported by a Department of Computing scholarship from Imperial College London, and a Qualcomm Innovation Fellowship. G.B. is funded by the CIAR project at IRT Saint-Exupéry. S.Z. was partially funded by the EPSRC Fellowship DEFORM: Large Scale Shape Analysis of Deformable Models of Humans (EP/S010203/1) and an Amazon AWS Machine Learning Research Award. The authors are grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support. This work was granted access to the HPC resources of IDRIS under the allocation 2020-AD011011311R1 made by GENCI.

A. DGCNN and ModelNet40

In this appendix, we provide details of the DGCNN model and of the ModelNet40 dataset ommitted from the main text for brevity.

ModelNet40 classification The ModelNet40 dataset [58] contains 12311 shapes representing 3D CAD models of man-made objects pertaining to 40 categories. We follow the experimental setting of [55] and [9]. We keep 9843 shapes for training and 2468 for testing. We uniformly sample 1024 points on mesh faces weighted by surface area and normalize the resulting point clouds in the unit sphere. The original meshes are discarded. Only the 3D cartesian coordinates (x, y, z) of the points are used as input. We use the same data augmentation techniques (random scaling and perturbations) as [55] and base our implementation on the author's public code[2]. We report the overall accuracy as the model score.

Model architecture All DGCNN models use 4 EdgeConv (or BinEdgeConv or XorEdgeConv) layers with 64, 64, 128, and 256 output channels and no spatial transformer networks. According to the architecture of [55], the output of the four graph convolution layers are concatenated and transformed to node embeddings of dimension 1024. We use both global average pooling and global max pooling to obtain graph embeddings from all node embeddings; the resulting features are concatenated and fed to a three layer MLP classifier with output dimensions 512, 256, and 40 (the number of classes in the dataset). We use dropout with probability $p = 0.5$.

B. Low-level implementation

This appendix provides further details on the low-level implementation and memory cost of our models.

B.1. Parameter counts

We report the counts of binary and floating-point parameters for the baseline DGCNN and our binary models (stage 3) in Table 5.

Model	FP32 Param.	Bin. param.	Total param.
Baseline	1,812,648	0	1,812,648
BF1	11,064	1,804,672	1,815,736
BF2	11,064	1,804,672	1,815,736
RF	15,243	1,804,672	1,819,915

Table 5. Number of parameters given by torchsummaryX. Separated into FP and binary operations. 99.39% of the parameters are binary for **BF1** and **BF2**, 99.16% of the parameters are binary for **RF**.

As can be seen in Table 5, our binarization procedure introduces a few extra parameters, but over 99% of the network parameters are binary.

[2] https://github.com/WangYueFt/dgcnn/tree/master/pytorch

B.2. Profiling and optimization of DGCNN

In order to obtain the data from Section 5.1 of the main paper, we convert our models with the LARQ converter and benchmark them using the LCE benchmark utility.

The pairwise Hamming distance is naively implemented as a matrix multiplication operation (Eq. 21 of the main text), and we obtain the profiler data in Table 6, where we have highlighted the nodes used by that operation. However, not all nodes of these types belong to the three pairwise distances calculations. We thus provide in Table 7 the complete profiler output for only one distance calculation in binary space, of which there are three in the DGCNN models.

Node Type	Avg. ms	Avg %	Times called
TOPK_V2	488.007	22.18%	4
CONCATENATION	384.707	17.485%	6
FULLY_CONNECTED	171.175	7.77994%	32
PRELU	143.086	6.50329%	7
TILE	136.443	6.20137%	4
LceBconv2d	127.371	5.78904%	6
MAX_POOL_2D	122.743	5.5787%	5
MUL	105.993	4.81741%	11
SUB	92.382	4.19878%	4
LceQuantize	91.168	4.14361%	10
NEG	78.453	3.56571%	4
PACK	56.301	2.55889%	4
GATHER	55.989	2.54471%	4
CONV_2D	39.096	1.77692%	2
RESHAPE	35.091	1.59489%	82
ADD	28.557	1.29792%	6
TRANSPOSE	23.829	1.08303%	36
AVERAGE_POOL_2D	8.071	0.366829%	1
SLICE	5.278	0.239886%	64
LceDequantize	5.174	0.235159%	4
SUM	1.132	0.0514497%	1
SQUARE	0.153	0.00695389%	1
SOFTMAX	0.01	0.000454502%	1

Table 6. LCE Profiler data for "BDGCNN BF H", summary by node types. In red: nodes that appear in Matmul op which can be rewritten as NEON operations for Hamming distance.

These operations account for 24% of the network's run time. Thus, a speed-up of 32x of these operations would reduce them to around 1% of the network's run time, which is negligible.

While we did not have an optimized version integrated with the LARQ runtime at the time of writing, optimizing the pairwise Hamming distance computation in binary space with ARM NEON (SIMD) operations is quite simple, since it can be implemented as popcount($x$xor$y$). On bit-packed 64-bit data (conversion handled by LCE), with feature vectors of dimension 64, this can be written as:

```
#include "arm_neon.h"

// input data in feats
int8_t n_outs = npoints*(npoints-1)/2
int8_t* out = malloc(n_outs*sizeof(int8_t));
for(int i = 0; i < npoints; i++) {

    // load first feature
    uint32x2_t a = vld1_u32(feats + 8*i);
```

Node type	Avg. time	Avg? %	Operation name
TRANSPOSE	5.91618	0.268874%	[bin_dgcnn_b_f1_h/MatMul_315]:208
SLICE	0.45752	0.020793%	[bin_dgcnn_b_f1_h/MatMul_316]:209
RESHAPE	0.43792	0.0199023%	[bin_dgcnn_b_f1_h/MatMul_317]:210
SLICE	0.14872	0.00675892%	[bin_dgcnn_b_f1_h/MatMul_318]:211
RESHAPE	0.22018	0.0100066%	[bin_dgcnn_b_f1_h/MatMul_319]:212
SLICE	0.16006	0.0072743%	[bin_dgcnn_b_f1_h/MatMul_320]:213
RESHAPE	0.22242	0.0101084%	[bin_dgcnn_b_f1_h/MatMul_321]:214
SLICE	0.16268	0.00739337%	[bin_dgcnn_b_f1_h/MatMul_322]:215
RESHAPE	0.21126	0.0096012%	[bin_dgcnn_b_f1_h/MatMul_323]:216
SLICE	0.15406	0.00700161%	[bin_dgcnn_b_f1_h/MatMul_324]:217
RESHAPE	0.20686	0.00940123%	[bin_dgcnn_b_f1_h/MatMul_325]:218
SLICE	0.1494	0.00678983%	[bin_dgcnn_b_f1_h/MatMul_326]:219
RESHAPE	0.21348	0.00970209%	[bin_dgcnn_b_f1_h/MatMul_327]:220
SLICE	0.15514	0.00705069%	[bin_dgcnn_b_f1_h/MatMul_328]:221
RESHAPE	0.2117	0.00962119%	[bin_dgcnn_b_f1_h/MatMul_329]:222
SLICE	0.15154	0.00688708%	[bin_dgcnn_b_f1_h/MatMul_330]:223
RESHAPE	0.17318	0.00787056%	[bin_dgcnn_b_f1_h/MatMul_331]:224
SLICE	0.15244	0.00692799%	[bin_dgcnn_b_f1_h/MatMul_332]:225
RESHAPE	0.16864	0.00766423%	[bin_dgcnn_b_f1_h/MatMul_333]:226
SLICE	0.15614	0.00709614%	[bin_dgcnn_b_f1_h/MatMul_334]:227
RESHAPE	0.1736	0.00788965%	[bin_dgcnn_b_f1_h/MatMul_335]:228
SLICE	0.15118	0.00687072%	[bin_dgcnn_b_f1_h/MatMul_336]:229
RESHAPE	0.17086	0.00776513%	[bin_dgcnn_b_f1_h/MatMul_337]:230
SLICE	0.149	0.00677165%	[bin_dgcnn_b_f1_h/MatMul_338]:231
RESHAPE	0.16924	0.0076915%	[bin_dgcnn_b_f1_h/MatMul_339]:232
SLICE	0.1505	0.00683982%	[bin_dgcnn_b_f1_h/MatMul_340]:233
RESHAPE	0.16894	0.00767787%	[bin_dgcnn_b_f1_h/MatMul_341]:234
SLICE	0.14994	0.00681437%	[bin_dgcnn_b_f1_h/MatMul_342]:235
RESHAPE	0.17058	0.0077524%	[bin_dgcnn_b_f1_h/MatMul_343]:236
SLICE	0.15018	0.00682528%	[bin_dgcnn_b_f1_h/MatMul_344]:237
RESHAPE	0.16996	0.00772422%	[bin_dgcnn_b_f1_h/MatMul_345]:238
SLICE	0.1496	0.00679892%	[bin_dgcnn_b_f1_h/MatMul_346]:239
RESHAPE	0.17112	0.00777694%	[bin_dgcnn_b_f1_h/MatMul_347]:240
TRANSPOSE	0.41792	0.0189933%	[bin_dgcnn_b_f1_h/MatMul_348]:241
FULLY_CONNECTED	8.78396	0.399207%	[bin_dgcnn_b_f1_h/MatMul_349]:242
TRANSPOSE	0.72016	0.0327293%	[bin_dgcnn_b_f1_h/MatMul_350]:243
FULLY_CONNECTED	8.64452	0.39287%	[bin_dgcnn_b_f1_h/MatMul_351]:244
TRANSPOSE	0.71804	0.032633%	[bin_dgcnn_b_f1_h/MatMul_352]:245
FULLY_CONNECTED	8.63224	0.392312%	[bin_dgcnn_b_f1_h/MatMul_353]:246
TRANSPOSE	0.72162	0.0327957%	[bin_dgcnn_b_f1_h/MatMul_354]:247
FULLY_CONNECTED	8.62624	0.392039%	[bin_dgcnn_b_f1_h/MatMul_355]:248
TRANSPOSE	0.68654	0.0312014%	[bin_dgcnn_b_f1_h/MatMul_356]:249
FULLY_CONNECTED	8.6722	0.394128%	[bin_dgcnn_b_f1_h/MatMul_357]:250
TRANSPOSE	0.69886	0.0317613%	[bin_dgcnn_b_f1_h/MatMul_358]:251
FULLY_CONNECTED	8.6892	0.394901%	[bin_dgcnn_b_f1_h/MatMul_359]:252
TRANSPOSE	0.71076	0.0323021%	[bin_dgcnn_b_f1_h/MatMul_360]:253
FULLY_CONNECTED	8.70248	0.395504%	[bin_dgcnn_b_f1_h/MatMul_361]:254
TRANSPOSE	0.71256	0.0323839%	[bin_dgcnn_b_f1_h/MatMul_362]:255
FULLY_CONNECTED	8.76456	0.398326%	[bin_dgcnn_b_f1_h/MatMul_363]:256
PACK	13.822	0.628173%	[bin_dgcnn_b_f1_h/MatMul_364]:257
SUB	29.9335	1.3604%	[bin_dgcnn_b_f1_h/sub_3;bin_dgcnn_b_f1_h/MatMul_3;b]:258

Table 7. LCE Profiler data for a single Hamming distance computation as a matrix multiplication, in "BDGCNN BF H".

```
    for(int j = i; j < npoints; j++) {

        //load second feature
        uint32x2_t b = vld1_u32(feats + 8*j);

        b = veor_u32(a, b); // XOR op

        // popcount op
        int8x8_t c = vreinterpret_u32_s8(b);
        c = vcnt_s8(c);

        // reduce to single number
        // by adding as a tree
        int64x1_t res;
        res = vpaddl_s32(vpaddl_s16(vpaddl_s8(c)));

        //store the output (last 8 bits)
        int8x8_t res8 = vreinterpret_s64_s8(res);
        out[j + npoints*j] = vget_lane_s8(res8, 7);
    }
}
```

Listing 1. Implementation of pairwise Hamming distance in ARM NEON instrinsics (for readability). Note that this code actually treats 64 features at a time and could thus provide a 64x speedup (or more by grouping loads and writes with vld4). We use 32x

as a conservative estimate since we couldn't account for LCE's bit-packed conversion.

"TopK" operations account for 22% of the runtime and we view them as incompressible in our simulation (Table 6). It is possible that they could be written in NEON as well, however, this optimization is not as trivial as the Hamming distance one. Remaining operations, such as "Concatenation", cannot be optimized further.

Contrary to simpler GNNs such as GCN, DGCNN is quite computationally intensive and involves a variety of operations on top of simple dot products, which makes it an interesting challenge for binarization, and illustrate that for complex graph neural networks more efforts are required, such as redefining suitable edge messages for binary graph features, or speeding-up pairwise distances computations, as done in this work. The inherent complexity also limits the attainable speedups from binarization, as shown by the large portion of the runtime taken by memory operations (concatenation) and top-k.

C. Details regarding GraphSAGE

In all experiments, the architecture used is identical to that used as a baseline by the OGB team. We report the accuracy following verbatim the experimental procedure of the OGB benchmark, using the suitable provided evaluators and dataset splits. Due to the very large number of edges in the dataset, we were unable to implement LSP in a sufficiently scalable manner (although the forward pass of the similarity computation can be implemented efficiently, the gradient of the similarity with respect to the node features is a tensor of size $|\mathcal{E}| \times |\mathcal{V}| \times D$ where $|\mathcal{E}|$ is the number of edges in the graph, $|\mathcal{V}|$ the number of nodes, and D the dimension of the features. Although the tensor is sparse, Pytorch currently did not have sufficient support of sparse tensors for gradients. We therefore chose not to include the results in the main text. We report the results of our binary GraphSAGE models, against two floating-point baselines: GraphSAGE and GCN.

D. Balance functions

For completeness, we also report the results at stage 2 of the multi-stage distillation scheme in Table 8. It is apparent that the additional operations degraded the performance not only for the full-binary models of stage 3, but also for the models for which all inputs are binary but weights are real.

E. Table of mathematical operators

References

[1] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *ICLR*, 2021. 7

[2] Gaétan Bahl, Lionel Daniel, Matthieu Moretti, and Florent Lafarge. Low-power neural networks for semantic segmentation of satellite images. In *ICCV Workshops*, 2019. 1

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013. 3

[4] Alexander Bronstein, Michael Bronstein, and Ron Kimmel. *Numerical Geometry of Non-Rigid Shapes*. Springer Publishing Company, Incorporated, 1 edition, 2008. 1

[5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4), 2017. 1

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. *ICLR*, 2014. 2

[7] Adrian Bulat and Georgios Tzimiropoulos. XNOR-Net++: Improved binary neural networks. In *BMVC*, 2019. 2, 3

[8] Sergio Casas, Cole Gulino, Renjie Liao, and R. Urtasun. Spagnn: Spatially-aware graph neural networks for relational behavior forecasting from sensor data. *International Conference on Robotics and Automation (ICRA)*, 2020. 1

[9] R Qi Charles, Hao Su, Mo Kaichun, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*, 2017. 9

[10] Wei Lin Chiang, Yang Li, Xuanqing Liu, Samy Bengio, Si Si, and Cho Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. 2

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *NeurIPS 29*. 2016. 2

[12] Sybil Derrible and Christopher Kennedy. Applications of graph theory and network science to transit network design. *Transport reviews*, 31(4), 2011. 1

[13] Matthias Fey and Jan E Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 8

[14] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Muller. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. *CVPR*, 2018. 2

[15] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020. 2

[16] Lukas Geiger and Plumerai Team. Larq: An Open-Source Library for Training Binarized Neural Networks. *Journal of Open Source Software*, 5(45), 2020. 7

[17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017. 2

[18] Shunwang Gong, Mehdi Bahri, Michael M. Bronstein, and Stefanos Zafeiriou. Geometrically Principled Connections in Graph Neural Networks. In *CVPR*, 2020. 1

[19] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv:1412.6115*, 2014. 2

[20] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in Graph domains. *International Joint Conference on Neural Networks*, 2(May 2014), 2005. 2

[21] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017. 2, 3, 7

[22] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. 2, 3

[23] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020. 7, 8

[24] Kexin Huang, Cao Xiao, Lucas M Glass, Marinka Zitnik, and Jimeng Sun. Skipgnn: predicting molecular interactions with skip-graph networks. *Scientific reports*, 10(1), 2020. 1

[25] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks. In D Lee, M Sugiyama, U Luxburg, I Guyon, and R Garnett, editors, *NeurIPS*, 2016. 2

[26] Anees Kazi, Luca Cosmo, Nassir Navab, and Michael Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *arXiv e-prints*, 2020. 3

[27] Abdullah Khanfor, Amal Nammouchi, Hakim Ghazzai, Ye Yang, Mohammad R Haider, and Yehia Massoud. Graph neural networks-based clustering for social internet of things. In *International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020. 1

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014. 8

[29] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 2, 3

[30] H Lai, Y Pan, S Liu, Z Weng, and J Yin. Improved Search in Hamming Space Using Deep Multi-Index Hashing. *IEEE T-PAMI*, 29(9), 2019. 5

[31] Seunghyun Lee and Byung Cheol Song. Graph-based knowledge distillation by multi-head attention network. In *BMVC*, 2019. 2

[32] Edgar Lemaire, Matthieu Moretti, Lionel Daniel, Beno\^\it Miramond, Philippe Millet, Frederic Feresin, and Sébastien Bilavarn. An FPGA-based Hybrid Neural Network accelerator for embedded satellite image classification. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020. 1

[33] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *ICCV*, 2019. 1

[34] Wei Liu, Jun Wang, Rongrong Ji, Yu Gang Jiang, and Shih Fu Chang. Supervised hashing with kernels. In *CVPR*, 2012. 5

[35] Sagar Maheshwari Marinka Zitnik, Rok Sosič and Jure Leskovec. Biosnap datasets: Stanford biomedical network dataset collection. http://snap.stanford.edu/biodata, Aug. 2018. 2

[36] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. In *ICLR*, 2020. 2, 3

[37] Batul J Mirza, Benjamin J Keller, and Naren Ramakrishnan. Studying recommendation algorithms by graph analysis. *Journal of intelligent information systems*, 20(2), 2003. 1

[38] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolá, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. *CVPR*, 2017. 2

[39] Jan Motl and Oliver Schulte. The ctu prague relational learning repository. *arXiv:1511.03086*, 2015. 3, 7

[40] Walter Nelson, Marinka Zitnik, Bo Wang, Jure Leskovec, Anna Goldenberg, and Roded Sharan. To Embed or Not: Network Embedding as a Paradigm in Computational Biology. *Frontiers in Genetics*, 10, 2019. 1

[41] Mohammad Norouzi and David J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, 2011. 5

[42] Mohammad Norouzi, David J. Fleet, and Ruslan Salakhutdinov. Hamming distance metric learning. In *NeurIPS*, 2012. 5

[43] Mohammad Norouzi, Ali Punjani, and David J. Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE T-PAMI*, 2014. 5

[44] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. In *CVPR*, 2019. 2

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *NeurIPS*. 2019. 8

[46] Haotong Qin, Zhongang Cai, Mingyuan Zhang, Yifu Ding, Haiyu Zhao, Shuai Yi, Xianglong Liu, and Hao Su. Bipointnet: Binary neural network for point clouds. In *ICLR*, 2021. 3, 5

[47] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net : ImageNet Classification Using Binary. *ECCV*, 2016. 2, 4, 14

[48] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. on Neural Networks*, 20(1), 2009. 2

[49] Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *ICCV*, 2019. 2

[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lió, and Yoshua Bengio. Graph Attention Networks. *ICLR*, 2018. 2, 3

[51] Nitika Verma, Edmond Boyer, and Jakob Verbeek. FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis. In *CVPR*, 2018. 2

[52] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, Xiangjian He, Yiguang Lin, and Xuemin Lin. Binarized graph neural network. *arXiv:2004.11147*, 2020. 3, 5, 7

[53] Junfu Wang, Yunhong Wang, Zhen Yang, Liang Yang, and Yuanfang Guo. Bi-gcn: Binary graph convolutional network. *arXiv:2010.07565*, 2020. 3, 7

[54] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph Cnn for learning on point clouds. *ACM Trans. on Graphics*, 2019. 2, 3, 5

[55] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. on Graphics*, 38(5), 2019. 8, 9

[56] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019. 2

[57] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. on Neural Networks and Learning Systems*, 2020. 2

[58] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015. 9

[59] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling Knowledge From Graph Convolutional Networks. In *CVPR*, 2020. 3

[60] Sergey Zagoruyko and Nikos Komodakis. Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer. In *ICLR*, 2017. 2

[61] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*, 2020. 2

[62] Weishan Zhang, Yafei Zhang, Liang Xu, Jiehan Zhou, Yan Liu, Mu Gu, Xin Liu, and Su Yang. Modeling iot equipment with graph neural networks. *IEEE Access*, PP, 2019. 1

[63] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160*, 2016. 2

Model	Stage	KNN	LSP	Global balance	Edge balance	Acc
BF2	2	H	-	-	Median	90.07
BF2	2	H	-	-	Mean	83.87
BF2	2	H	ℓ_2	Median		87.60
BF2	2	H	ℓ_2	Mean		89.47
Baseline BF2	2	H	ℓ_2	None		**91.57**

Table 8. Effect of additional balance functions on models with binary activations but floating-point weights. The performance of the baseline model suffers with the introduction of either mean or median centering prior to quantization.

Symbol	Name	Description
$\|\|.\|\|_H$	Hamming norm	Number of non-zero (or not -1) bits in a binary vector
$d(.,.)_H$	Hamming distance	Number of bits that differ between two binary vectors, equivalent to `popcount(xor())`
$\oplus$	Exclusive OR (XOR)	$1 \oplus 1 = -1 \oplus -1 = -1, \quad -1 \oplus 1 = 1 \oplus -1 = 1$
$\odot$	Hadamard product	Element-wise product between tensors
$\circledast$	Binary-real or Binary-binary dot product or convolution	Equivalent to `popcount(xnor())` (*i.e.* no multiplications) for binary tensors
$\otimes$	Outer product	
$\star$	Dot product or convolution	Denoted by $*$ in [47]
$\|\mathcal{X}\|$	Cardinal of a set $\mathcal{X}$	Number of elements in the set
$\mathbf{x}^{(l)}$	Feature maps at layer l	
$.\|\|.$	Concatenation	
$:=$	Definition	
$\mathbf{x}^{(l)}$	Element $\mathbf{x}$ at layer l	

Table 9. Table of the mathematical operators used in the manuscript.